



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|-------------|----------------------|---------------------|------------------|
| 10/598,332 | 03/09/2007 | Johan Eker | P19142-US3 | 2736 |
| 27045 | 7590 | 02/22/2012 | EXAMINER | |
| ERICSSON INC. 6300 LEGACY DRIVE M/S EVR 1-C-11 PLANO, TX 75024 | | | VU, TUAN A | |
| | | | ART UNIT | PAPER NUMBER |
| | | | 2193 | |
| | | | NOTIFICATION DATE | DELIVERY MODE |
| | | | 02/22/2012 | ELECTRONIC |

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

kara.coffman@ericsson.com
jennifer.hardin@ericsson.com
melissa.rhea@ericsson.com

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 12/16/11.

As indicated in Applicant's response, claims 19, 31 have been cancelled. Claims 17-18, 20-30, 32-42 are pending in the office action.

Double Patenting

2. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and, *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

3. Claims 17, 29, 41 are provisionally rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claim 30 of copending Application No. 12,064,073 (hereinafter '073) in view of APA (Admitted Prior Art: Specifications: pg. 1-2) further in view of Sweeney, USPN: 6,401,182 (Sweeney) and Miyawaki et al, USPubN: 2005/0157619 (Miyawaki)

Although the conflicting claims are not identical, they are not patentably distinct from each other because of the following observations. Following are but a few examples as to how the certain claims from the instant invention and from the above copending application are conflicting with each other.

Art Unit: 2193

As per instant claim 17, '073 claim 30 also recites 'performing optimization process to reduce differences between updated object code and one or more corresponding one of set of current object code', the updated object code as input to a linker for generation of 'an updated memory image ... into a storage medium ... having a current version of a computer program'; hence has recited a obvious variant to instant claim 17 regarding transforming ... source code into an updated program version to be stored in memory ... stored thereon a current version' and 'transforming ... steps of compiling ... resulting in ... object modules ... receiving representation of current version and performing optimization and performing a linking step'.

However, '073 does not recite 'current code version occupying first set of memory sectors... updated code version occupies a second set of memory sectors ..., Liu discloses optimization adapted to decrease the number of ... sectors in the second set ... updated code version that are different from ... of the first set memory section ... current code version. APA discloses updating of firmware or flash memory including upgrading of pages or memory sectors thereof, with need to adjust flash sectors (Specifications: pg. 1-2), the updating implicating a plurality of memory sectors. Based on the concept of optimizing with resulting size differences as recited in '073, it would have been obvious for one of ordinary skill in the art to implement '073 so that optimization is adapted to decrease number of store sectors in the current version, so that the updated version occupying the target memory would occupy a different set of sectors to those of the current version, whereby using optimization as shown in APA would write updated code to sectors with adjusting of space difference as needed.

Further, '073 does not recite 'generating feedback during linking ... recompiling ... subset of the ... modules based on the feedback ... resulting in a number of modified object modules,

Art Unit: 2193

and performing linking of ... modified object modules. '073 recite subjecting "updated object module as an input to a linker for generation of an updated memory image" or 'object code adapted to be linked ... to generate updated memory image'; and using "control data from a linker component", the 'control data includes a size constraint for the updated object code module', hence has taught a role played by linker generated data into generating of updated code. Similar to '073, real-time adaptive aspect of heap managers is taught in Sweeney for recompiling (Sweeney: col. 8 line 55 to col. 9 line 18) based on sectors of memory affected by update program (see Fig. 3-4) where header are to be padded to match a difference in sectors size (Fig. 16-19) whereas writing data into medium similar to flash/firmware write in APA is disclosed in Miyawaki with control type of overhead (similar to manager in Sweeney) so as to store meta-information on padding in regard to encountered discrepancies between size of optical memory sectors targeted for recording media as this padding is iteratively needed for successive medium writes; and it would have been obvious for one of ordinary skill in the art to implement the use of linker in '073 so that linker 'control data' would serve as feedback into the compiler for the compiler to address size difference (e.g. padding as in Miyawaki or Sweeney) and readjust updated code module accordingly and resubmit the modified code modules to further compilation (e.g. recompilation – as in Sweeney) coupled to further linking to yield the size controlled or readjusted based on the constraints fed from the linker; and this would enable size differences as feared in the sector adjusting by APA to be correctively adapted to yield a reduced memory image while respecting size constraints between sectors occupied by current version and sectors (as taught in Sweeney or Miyawaki) occupied by updated version.

As per instant claims 29 and 41, these claims recite the analogous subject matter of instant claim 17; hence would have been deemed an obvious variant to '073 claim 30 in view of the rationale set forth above.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. Claims 17, 23, 26-29, 35, 38-42 are rejected under 35 U.S.C. 103(a) as being unpatentable over Liu et al, USPubN: 2004/0068719(herein Liu) in view of in view of APA (Admitted Prior Art: Specifications: pg. 1-2) and/or Lohse et al, USPubN: 2003/0142556 (herein Lohse)

As per claim 17, Liu discloses a method for updating program code stored in a memory, the memory having a plurality of memory sectors, the method comprising the steps of:

transforming at least one updated source code module into an updated program code version to be stored in a memory (source code ... intermediate objects - para 0010-0015, pg. 2; step 400, step 414 – Fig. 4A; memory 204 – para 0039, 0041) wherein the memory has stored thereon a current program code version occupying a first set of the memory sectors of the memory (intermediate objects – para 0010, 0012-0013);

wherein the transforming step further comprises the steps of compiling the at least one source code module resulting in a number of object modules (Fig. 4A, 4B; intermediate objects and object files, real objects – para 0024-0025), receiving a representation of the current

Art Unit: 2193

program code version, and performing at least one optimization step adapted to decrease the number of memory sectors of the second set of memory sectors (e.g. para 0053-0054, pg. 6; short format procedure calls and references through the linkage tables – para 0031; extension bits ... sizes of section - para 0032 - Note: use of plug-ins to gather linkage table and size information adapted to transform global data into short data - see para 0038- for procedure linkage –see para 0051 -- reads on optimization to decrease number of memory sectors for an updated current version – see: *as much of short data as possible* – para 0053 – thereby yielding optimized final code – step 314, Fig. 3); and

wherein the performing at least one optimization step, further comprises the steps of generating feedback data during a linking step for linking the number of object modules (para 0028-0030; feed-back plug-in – para 0052; feed-back – para 0048 – Note: intermediate objects from compiler passed to a second phase linker **reads on** initial compilation - see para 0047 -- prior to final real objects being generated -- i.e. via recompilation -- using feed-back linkage tables information – see Fig. 4B,C), re-compiling at least a subset of the source code modules based on the feedback data and resulting in a number of modified object modules (e.g. *generate real objects, are linked together* – para 0048; para 0051-0052), and performing a linking step based on the number of modified object modules(e.g. *real objects ... are then be linked together* – para 0054, pg. 6).

Liu does not explicitly disclose: updated code version in memory in terms that *the memory has stored thereon a current program code version occupying a first set of the memory sectors of the memory, wherein the updated program code version occupies a second set of memory sectors when stored in the memory*; nor does Liu explicitly disclose optimization (to

Art Unit: 2193

decrease the number of memory sectors) in terms of second set of *memory sectors occupied by the updated code version that are different from the corresponding memory sectors of the first set of memory sectors occupied by the current program code version*

Liu discloses optimization to obviate size storage or execution delay for overusing long data (see para 0002-0004) in favor of short data allocation (see para 0008, pg. 1) where large amount of otherwise long data is allocated to short data (allocates linkage tables... allocates data as short data – para 0053-0054, pg. 6) using section size information (para 0032) gathered by the first-pass plug-in invocations (Fig. 4B, 4C) to convert global data related to intermediate and real object to transform subsequent to the first pass linker, using linkage tables (para 0038) for short data to be used in procedure linkage (para 0051) entails aggressive optimization via a second pass by which the real objects produced as executable program would occupy less memory sectors (para 0054) . Hence the concept of occupying a different memory space for a updated version of code (e.g. re-allocated short data) with respect to its original version (larger store of long data) is indicated; that is Liu **either discloses or would render obvious** updated program code version occupying a different memory sectors, based on the second pass and linker using *as much short data as possible* (para 0053-0054) than the first set of memory sectors occupied by the program version before the aggressive optimization, which also signify: *memory has stored thereon a current program code version occupying a first set of the memory sectors of the memory* (prior to compiler optimizing linker), *wherein the updated program code version occupies a second set of memory sectors when stored in the memory* (after the optimization by the linker feedback second pass). Regarding obviousness, following is the rationale.

Art Unit: 2193

Similar to memory of portable devices or ASIC system or firmware storage medium mentioned in Liu (see Liu: PDA, ASIC firmware medium – para 0039, 0041, 0044) **APA** discloses update of flash memories by vendors (Specifications: pg. 2, top), where embedded memory of electronic device are pages of entire sectors that can be flashed (Specifications: pg. 1, middle) such that, for example, sectors of a update flash memory can be stored at proximity of other sectors in the current memory space of the target device (Specifications: 2nd para pg. 2), which is corroborated in **Lohse** (see Lohse: para 0025-0028, pg. 2). Based on the above writing of update data into sectors of embedded devices where the update occupies other sectors (e.g. as by Lohse) than those taken by the current firmware or flash prior to the writing, it would have been obvious for one of ordinary skill in the art to implement the optimization of code (e.g. for microprocessor embedded devices including firmware/flash memory well-known in portable devices) with reduced size based on Liu's approach so that the code update would occupy sectors of memory (of said target microprocessor or embedded devices to be upgraded) and include the optimized code as in Liu (e.g. via a flash write operation or off-air update as in APA) such that the update sectors being flashed or written would be allocated in different memory space with respect to other sectors storing the original code version prior to its being upgraded. One would be motivated to do so because update code segregated in specific sectors not intermingling with the current version can be subjected to incidental adjustment or modification, and any such adjustment or initialization (e.g. operative only a newly allocated and different memory sectors (as in Lohse) not interfering with sectors storing current firmware or operational memory of a device – PDA, ASIC, embedded microprocessor as in Liu) would not affect functional operation state of the embedded device or any target processor receiving the update flash or firmware into

Art Unit: 2193

its memory; as any maintenance type of setback affecting device operational state (e.g. setback with operational downtime) as a result of reflashing devices are concerned taught in APA (see Specifications: pg. 1 last para)

As per claim 23, Liu discloses wherein the transforming step further comprises the step of controlling the optimization step by at least one optimization parameter (*symbol_iterator* – para 0038; *short_data_size* – para 0038; *size ... linkage tables* - para 0051 - Note: parameter obtained from plug in regarding iterated queries on nature or size of symbol or size of linkage tables reads on optimization parameter).

As per claim 26, Liu does not explicitly disclose
comprising generating a delta file representative of differences between the current program code version and the updated program code version.

APA discloses well-known practice of using delta-file in support for upgrading a client medium by OTA vendors (Specifications: pg. 1 bottom) whereas delta-file is also disclosed in Ren's approach (Ren: Fig. 1-2; para 0030-0032). It would have been obvious for one of ordinary skill in the art to implement the associated meta-information supporting Liu's optimization of source code space so that source memory address space being updated with updated objects would include a delta-file because processing the slightest differences between memory sectors implicated by the update would help deriving large patterns or minimal set therein, in order to adaptively implement the appropriate as-needed modification algorithms that would otherwise require extraneous optimization code resources, based on the delta file scanning approach by Ren.

Art Unit: 2193

As per claims 27-28, Liu (in view of APA and Lohse) discloses portable devices, such that according to the rationale in claim 17, wherein the memory is a flash memory, wherein the memory is a memory of a portable radio communications equipment.

As per claim 29, Liu discloses a data processing system for updating program code stored in a memory, the memory having a plurality of memory sectors, the data processing system comprising:

transformation means adapted to transform at least one updated source code module into an updated program code version to be stored in a memory (refer to claim 17);

the transformation means further having a compilation means adapted to compile the at least one source code module resulting in a number of object modules (refer to claim 17),

a reception means adapted to receive a representation of the current program code version, and a performance means adapted to perform at least one optimization operation to decrease the number of memory sectors of the second set of memory sectors (refer to claim 17); and

wherein the performance means adapted to perform the at least one optimization operation is further adapted to generate feedback data (refer to claim 17) while the number of object modules are being linked, re-compile at least a subset of the source code modules based on the feedback data resulting in a number of modified object modules (refer to claim 17), and

perform a linking operation based on the number of modified object modules (refer to claim 17).

Liu does not explicitly disclose updated version stored in memory in terms of said memory has stored thereon a current program code version occupying a first set of the memory

Art Unit: 2193

sectors of the memory, wherein the updated program code version occupies a second set of memory sectors when stored in the memory. Nor does Liu explicitly disclose second set of memory sectors occupied by the updated code version that are different from the corresponding memory sectors of the first set of memory sectors occupied by the current program code version.

But the above has been addressed in claim 17.

As per claim 35, refer to claim 23, accordingly.

As per claims 38, refer to claim 26, accordingly.

As per claims 39-40, refer to claims 27-28, accordingly

As per claim 41, Liu discloses (in view of APA and/or Lohse) a computer program product comprising program code means adapted to cause a data processing system to perform steps when the program code means are executed on the data processing system, the steps comprising:

transforming at least one updated source code module into an updated program code version to be stored in a memory, which memory has stored thereon a current program code version occupying a first set of the memory sectors of the memory, wherein the updated program code version occupies a second set of memory sectors when stored in the memory;

wherein the transforming step further comprises the steps of compiling the at least one source code module resulting in a number of object modules, receiving a representation of the current program code version, and

performing at least one optimization step adapted to decrease the number of memory sectors of the second set of memory sectors occupied by the updated code version that are

Art Unit: 2193

different from the corresponding memory sectors of the first set of memory sectors occupied by the current program code version; and

wherein performing the at least one optimization step, further comprises the steps of generating feedback data during a linking step for linking the number of object modules, re-compiling at least a subset of the source code modules based on the feedback data and resulting in a number of modified object modules, and

performing a linking step based on the number of modified object modules; all of which having been addressed in claim 17.

As per claim 42, Liu discloses wherein the computer program product comprises a linker module (Linker 108- Fig. 1).

6. Claims 18, 30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Liu et al, USPubN: 2004/0068719(herein Liu) in view of in view of APA (Admitted Prior Art: Specifications: pg. 1-2) and/or Lohse et al, USPubN: 2003/0142556 (herein Lohse) further in view of Ren et al, USPubN: 2004/0260734 (herein Ren)

As per claim 18, Liu does not explicitly disclose wherein the representation of the current program code version comprises at least one of a current image of the first set of memory sectors and a map file description of the current image of the first set of memory sectors.

Analogous to the same concept taught in APA's context of device upgrade following vendor/client upgrade paradigm (see instant Specifications: *delta-file* - pg.1, bottom) so as to modify a current memory image, **Ren** also discloses using delta-file in conjunction with a *map file* (Ren: para 0031; Fig. 2) the map file information provided by a compiler/linker engine being similar to the tabular linkage configuration generated by Liu's linker to match a symbol to

Art Unit: 2193

various preemptability factors like long access references or types for that symbol(see Liu: para 0051). It would have been obvious for one of ordinary skill in the art to implement the linkage type information in LIU so that this is structured as a map file as in Ren to support the delta-file memory upgrade as shown in APA in light of the optimization approach in Liu where optimized memory store can receive size reduced sectors for targeted image as disclosed in APA, the map file provided by a linker stage and enabling reference information useful to the final generation of optimized data to be written in the target image (of an embedded device as per the rationale in claim 17), including the start address and size of each symbol of a software image, with symbol examples including function and global variables (as in Ren), since function and global data (see Liu: Fig. 1) are subjected to Liu's approach to reduce large code size into short data space allocation.

As per claim 30, refer to rationale of claim 18.

7. Claims 20, 32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Liu et al, USPubN: 2004/0068719(herein Liu) in view of in view of APA (Admitted Prior Art: Specifications: pg. 1-2) and/or Lohse et al, USPubN: 2003/0142556 (herein Lohse) and Ren et al, USPubN: 2004/0260734 (herein Ren), further in view of Szewerenko et al, USPubN: 2001/0047512 (herein Szewerenko) and/or O'Boyle et al, "Feedback Assisted Iterative Compilation", May 2000, pp. 1-9 (herein OBoyle)

As per claim 20, Liu does not explicitly disclose

wherein the optimization step further comprises the step of *determining a layout of the object modules in memory; or wherein the optimization step further comprises the step of determining a layout of the object modules in memory.*

Liu discloses space related investigation via a scan over locations of affected procedure calls or symbols as a first pass (para 0031) and a second pass by the compiler analyzing syntax hierarchy to correlate symbols to preemptable set of objects (para 0049-0051); and according to size information as basis for feedback from the linker support, generation linkage tables (Fig. 4C) to effectuate reducing of space. Space reducing by way of reconfiguring sectors of the target memory/medium is disclosed in **Lohse** (refer to claim 18) such that, for example, correspondence between target locations or sectors addresses to be written to and current addresses of an memory image is disclosed as linker's provision of mapping information taught in **Ren** with laying out addresses of affected sectors (see Ren: para 0031). Analogous to investigation of code space or static tree analysis from above, **OBoyle** discloses recompilation approach based on profile feedback (OBoyle: Fig. 2, pg. 3) in which space exploration uses a 2-D grid where regions thereof are investigated to enable re-allocation, parameter transformation leading to memory/block reordering (see OBoyle: sec 3.1.1, 3.1.2; 3.2, 3.2.1). Similar to the linker feedback by Liu, **Szewerenko** further discloses a optimization compiler with linker feedback (Szewerenko: para 0065) where prior to writing to memory sectors of target processor, *layout information* is provided (Szewerenko: para 0070; para 0030, 0043) where layout is determined via a Gui to facilitate allocation of object modules(Szewerenko: para 0054; para 0043) and facilitating a linking process, which is analogous approach to Liu's use of linker tables, to allocate blocks of code to memory without confidence checks. Based on the address, sectors delimitation and memory locations being specific space-related information useful in support writing a target memory as shown in Lohse or Ren, it would have been obvious for one of ordinary skill in the art to implement investigation of code space – as in Liu – in conjunction

Art Unit: 2193

with employing concurrent linker support (using results of such space analysis), so that layout information – as per Szewerenko - regarding sector or regions of memory space to be optimized or rewritten with updated object/data is provided in support of the linker, based on Liu's approach, because layout of memory space targeted for optimization as shown in Oboyle or Szewerenko, would enable a developer using the linker functionality to provide proper reallocation of blocks as shown above, or to provide appropriate parameter transformation as in OBoyle in order to support dynamic memory re-ordering prior to writing to a target medium, as proffered in APA/Lohse or Ren, thereby support space restraint aspect of embedded memory or flash storage of space-restraints devices like PDA or portable devices.

As per claim 32, refer to claim 20, accordingly.

8. Claims 21-22, 33-34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Liu et al, USPubN: 2004/0068719(herein Liu) in view of in view of APA (Admitted Prior Art: Specifications: pg. 1-2) or Lohse et al, USPubN: 2003/0142556 (herein Lohse) and Ren et al, USPubN: 2004/0260734 (herein Ren), further in view of Szewerenko et al, USPubN: 2001/0047512 (herein Szewerenko) and/or O'Boyle et al, "Feedback Assisted Iterative Compilation", May 2000, pp. 1-9 (herein OBoyle) and Sweeney, USPN: 6,401,182 (Sweeney)

As per claim 21, Liu does not explicitly disclose:

wherein determining the layout of the object modules in memory further comprises the steps of: *detecting an updated first object module having a different size than a corresponding first current object module*, and an updated second object module equal to a corresponding second current object module, which *updated second object module has a base address larger than the base address of the updated first object module*; and *padding the detected updated first*

Art Unit: 2193

object module with a predetermined memory content of a predetermined padding size resulting in a padded updated first object module; wherein the padding size is selected to cause the base address of the updated second object module to be equal to the base address of the corresponding second current object module.

The determining of layout has been addressed in claim 20. The concept of upgrading a memory space with upgraded result occupying less size than the current code prior to upgrade entails determining that either updated first object module having a different size than a corresponding first current object module, and either an updated second object module equal to a corresponding second current object module. OBoyle discloses such determination in terms of addressing size difference via strategies including space grid exploration and array padding (sec 3.2, pg. 5) of identified portion of the target code which require padding (e.g. OBoyle: array padding – sec 3.3.1, 3.3.2, 3.3.3) and the concept of padding based on difference in size is disclosed in the heap manager method by **Sweeney**; where Sweeney discloses additional information to mark how padding would be imparted when size difference between original space and modified code space is detected (Sweeney: Fig. 14-16) where Sweeney's heap manager is implemented with support for previous code space alignment analysis (set forth at compile time) similar to pre-compiler plug-ins approach taught in Liu's linker assist, the padding to equalize array structure size when size blocks are determined to be occupied unequally (col. 7 line 7 to col. 8 line 10). Based on use of map file and layout information to help allocate blocks as set forth in claim 20 with teachings from Ren and Szewerenko, the address to write a upgrade object being of size different from a current size of a memory based on the concept of mapping as above entails that first/second current object with base address larger than the base address of

Art Unit: 2193

the updated first/second object module, suggesting that the padding results in a padded updated first object module; where the padding size is selected to cause the base address of the updated second object module to be equal to the base address of the corresponding second current object module. Based on the well-known practice to use padding to overcome address size difference between locations of current data space in view of a intended modified data space, where static support of code space would be needed as shown above (e.g. Liu, Ren, Szewerenco, OBoyle or Sweeney), it would have been obvious for one of ordinary skill in the art to implement update of memory space in Liu using support of compiler's static layout or target space analysis such that where size difference is detected based on address at which to write updated object into a current memory space, padding would be implemented for specific code data (e.g. array structures), based on early profiling or layout/alignment analysis; because padding implementation as set forth in OBoyle or Sweeney, can equalize any detection of address discrepancy (i.e. *cause the base address of the updated first or second object module to be equal to the base address of the corresponding first or second current object module*) as set forth by means of early analysis or exploration of code space to be replaced with updated objects; averting thereby address discrepancies resulting in code execution memory conflict which would defeat the purpose of optimizing as set forth in Liu.

As per claim 22, Liu does not explicitly disclose detecting an updated first object module that is larger than a corresponding first current object module; moving a predetermined part of the updated first object module to a different memory sector resulting in a reduced updated first object module and a moved part of the updated first object module; and inserting a reference to the moved part of the updated first object module in the reduced first updated

Art Unit: 2193

memory sector. But the concept of implementing a *padding* as reference to moving code into a address space which entail size differences between updated object module and initial object module would fall under the same ambit as providing padding as addressed in claim 21 from above; hence moving of upgraded module to a *different memory sector resulting in a reduced updated first object module and a moved part of the updated first object module; and inserting a reference to the moved part of the updated first object module in the reduced first updated memory sector* would have been obvious for the same reasons as set forth thereon.

As per claims 33-34, refer to claims 21-22, accordingly.

9. Claims 24-25, 36-37 are rejected under 35 U.S.C. 103(a) as being unpatentable over Liu et al, USPubN: 2004/0068719(herein Liu) in view of in view of APA (Admitted Prior Art: Specifications: pg. 1-2) or Lohse et al, USPubN: 2003/0142556 (herein Lohse) further in view of O'Boyle et al, "Feedback Assisted Iterative Compilation", May 2000, pp. 1-9 (herein OBoyle) and Peyton, Jr et al, USPN: 5,920,723 (herein Peyton)

As per claims 24-25, Liu does not explicitly disclose

wherein the at least one optimization parameter includes a parameter determining a maximum allowed increase in size caused by the optimization step, wherein the at least one optimization parameter includes a parameter determining a maximum allowed number of references introduced by the optimization step.

Analogous to a limit of all defined object data being invoked by Liu's use of query plug-in to enumerate all the symbol structures (para 0038, pg. 4), **OBoyle** disclose exploration of code space in evaluating 2-grid dimensions representing such space, and this entail a maximum limit established by each dimension (see OBoyle: sec 3.1.2) where Boyle disclose tree-based

Art Unit: 2193

transformation using partition approaches associated with cost for the strategy used therewith, where cost entails limit in resource, each partition as budget for which the transformation algorithm would proceed until the budget resources are exhausted (OBoyle: sec 3.2, 3.2.1 pg. 5); hence optimization using a maximum limit associated with a strategy is disclosed. Further, **Peyton** discloses optimization (e.g. for inlining) on source code units organized for benefit evaluation via graph analysis similar to OBoyle or Liu (see Liu: syntax tree – para 0049) where a benefit value associating call sites and cost derived from the sites (col. 6 line 64 to col. 7 line 12) is more preferable or not based on a limit to number of code growth at a call site, associating therewith limits available memory to the code.

It would have been obvious for one of ordinary skill in the art to implement the size queries and symbol enumeration in Liu so that maximum limit of resources associated with a partitioned transformation algorithm support a best budget approach as shown in OBoyle selection of strategies, or a maximum code expansion at a given analyzed call sites would be input in support for cost-based selection of a best benefit parameter as in Peyton; because establishing a prior limit to how much code space or resources availability, -- each set as optimization parameter inputted in the determination of various code transformation approaches (e.g. maximum number of symbol references therein can be used or maximum resources – as in Peyton or OBoyle) without exceeding a unacceptable cost/benefit --- can support selection of best budget-oriented optimization strategy or transformation algorithm in which only a maximum code space is available for further expansion (see OBoyle or Peyton, respectively), the size and number of references delimited by that size not to demise the benefit of a given algorithm or strategy to transform a code space under analysis as purported in Liu's approach.

As per claims 36-37, refer to claims 24-25, accordingly.

Response to Arguments

10. Applicant's arguments filed 12/16/11 have been fully considered but they are not persuasive. Following are the Examiner's observation in regard thereto.

(A) Applicants have submitted re-characterizing a greater amount of data to "short data" as in Liu's approach is markedly different from the recited optimization step 'adapted to decrease the number of memory sectors ... occupied by the updated version that are different from the corresponding ... sectors of the first set of ... sectors occupied by the current ... code version" (Applicant's Remarks pg. 10, bottom, pg. 11 top). The process based on two passes by Liu enable linkage of procedure and associated data (in a initial translated code) that have been collected and analyzed in terms of size-related information from a first pass – feed-back phase using plug-in invocations – to be revisited by way of a second pass and to be translated code having "short data" as much as possible, as opposed to unsorted objects (global data, intermediate data, real data, without discrimination as to a specific size restraints) collected prior to invoking the plug-ins. Hence, the 2-pass linker approach in Liu implements a form of final code optimization, the optimization technique "adapted" to occupy less memory sectors by the translated second code version after the second phase (e.g. the resulting effect in form of less memory sectors used to store the final translated procedures), the final code version translation achieved mostly from *short data* based on the input fed as "short data" estimates from the first pass (i.e. via plug-in operating based on criteria enforcing a limit or range being determined during the feed-back pass) by which global objects, intermediate objects can be marked as "short data" for use by the last linker pass. The claim does not contain compelling method

Art Unit: 2193

implementation details that would depict a particular memory sector size occupying that would clearly read away from the above code size optimization by Liu 2-linker passes. Nor does the claim contain effect that upgraded code would achieve reduction of memory notably for the process of re-writing expressed in terms of *less number of re-writes*, as alleged in the argument (Applicant's Remarks pg. 11, top). Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the reference. Besides, one cannot see non-obviousness of the claim when the argument appears to dissect one reference (Liu) and leave out any other references actually employed jointly with the main reference (Liu) in establishing the 103 rationale of obviousness.

(B) Applicants have submitted that Ren fails to remedy to the deficiency of the combination of Liu/APA/Lohse for meeting or rendering obvious claims 18, 30; that Szeverencko and OBoyle fail to remedy to the combination of Liu/APA/Lohse for fulfilling claims 19-20, 31-33. Nor can Sweeny in view of all the above references fail to teach or suggest the features of claims 21-22, 33-34 (Applicant's Remarks pg. 11 bottom half). There is no true prima facie case of rebut in the mere alleging that a certain combination of references fail to teach a claim and/or that one additional reference fail to remedy to the deficiency of that combination; as one cannot see non-obviousness (in regard to any claim in question) when no "proof to the contrary" is provided for demising any prongs of the 103 rationale that has been actually executed on basis of the cited references and specific teachings thereof.

(C) Applicants have submitted that OBoyle and Peyton fail to remedy to the combination of Liu/APA/Lohse for meeting or rendering obvious claims 24-25, 36-37 (Applicant's Remarks pg.

Art Unit: 2193

12) This allegation for patentability without showing of facts is non-persuasive for the same reasons as set forth in sections A to B from above.

The claims stand rejected as set forth in the Office Action.

Conclusion

11. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before

Art Unit: 2193

using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

February 15, 2012